

## An algorithm for the Tutte polynomials of graphs of bounded treewidth

Artur Andrzejak\*

*ETH Zürich, Departement Informatik, ETH Zentrum, IFW, 8092 Zürich, Switzerland*

Received 14 December 1995; revised 26 February 1998; accepted 9 March 1998

---

### Abstract

Let  $k$  be a fixed, positive integer. We give an algorithm which computes the Tutte polynomial of any graph  $G$  of treewidth at most  $k$  in time  $O(n^{2+7 \log_2 c})$ , where  $c$  is twice the number of partitions of a set with  $3k + 3$  elements and  $n$  the number of vertices of  $G$ . © 1998 Elsevier Science B.V. All rights reserved

*Keywords:* Tutte polynomial; Computational complexity; Bounded treewidth

---

### 1. Introduction

Triggered by the paper of Jaeger et al. [13], the computational complexity of the Tutte polynomial has received a lot of attention. It was shown in [13] that for fixed values of  $x$  and  $y$  evaluating the Tutte polynomial  $t(M; x, y)$  of a matroid  $M$  is  $\#P$ -hard. This result holds unless the point  $(x, y)$  satisfies  $(x - 1)(y - 1) = 1$  or is one of 8 special points. Later Vertigan showed in [18] that a similar result holds for the Tutte polynomial of a planar graph.

The applications of the Tutte polynomial obtained by specialization range widely. They include such quantities as the chromatic and the flow polynomial of a graph, the (all terminal) reliability of a network and the Jones and Kauffman bracket polynomials of an alternating link [11, 13, 19]. Most of the quantities obtained by evaluating the Tutte polynomial (especially all listed above) are  $\#P$ -hard for the class of planar graphs.

Therefore, it is interesting to investigate for which classes of graphs the computation of the Tutte polynomial can be done in polynomial time. As shown in [17], one such class is the class of series-parallel networks.

---

\* E-mail: artur@inf.ethz.ch.

In this paper we show that for any fixed integer  $k$  the Tutte polynomial of a graph with treewidth at most  $k$  can be computed in polynomial time. To achieve this, we apply to an input graph  $G$  algorithms of Bodlaender and Hagerup [9,7]. As a result, we learn in linear time whether  $G$  has treewidth at most  $k$  and if so, we obtain a tree decomposition of  $G$  with certain properties. Then we apply to this tree decomposition an algorithm which computes the Tutte polynomial of  $G$  in time  $O(n^{2+7 \log_2 c})$ , where  $c$  is twice the number of partitions of a set with  $3k+3$  elements and  $n$  the number of vertices of  $G$ .

Noble [16] has independently obtained a linear time algorithm for computing the Tutte polynomial of  $G$  when its embedding in a  $k$ -tree is given. While his approach is close to the methods in [5], we give here a treedecomposition-guided algorithm which uses the notion of splitting formulas [15,2].

The classes of graphs of treewidth bounded by an integer  $k$  (or, equivalently, the classes of partial  $k$ -trees) are well studied. They include such graphs as series-parallel networks ( $k=2$ ), chordal graphs with maximal clique size  $k+1$  and interval graphs with maximal clique size  $k+1$  (see [8] for a survey). A number of problems being NP-hard in general turn out to be polynomial-time (or even linear-time) solvable for graphs of bounded treewidth. These problems include HAMILTONIAN CIRCUIT, CHROMATIC NUMBER, VERTEX COVER and many more (see [4,5]). Our result expands this class of problems.

## 2. Definitions and key ideas

Our definitions are standard and follow [1,19,9]. Let  $G$  be a graph possibly with loops and parallel edges.  $V(G)$  denotes the set of vertices of  $G$  and  $E(G)$  the set of edges.  $c(G)$  denotes the number of connected components of  $G$ . We put  $n = |V(G)|$ .

We use the following convention. If two vertices  $u, v \in V(G)$  have been identified to a vertex  $u'$ , then both symbols  $u$  and  $v$  refer to  $u'$ . This convention is also applied if we identify a set of vertices of  $G$  to a single vertex.

For  $e \in E(G)$ ,  $e = \{u, v\}$ , the *contraction*  $G/e$  is the following graph  $G'$ .  $E(G') = E(G) - \{e\}$ ,  $V(G') = (V(G) - \{u, v\}) \cup \{u'\}$ , where  $u$  and  $v$  are identified to a vertex  $u'$  in  $G'$ . For a set  $T \subseteq E(G)$ ,  $T = \{e_1, \dots, e_i\}$  for some  $i > 1$  we define the contraction  $G/T$  as  $((G/e_1)/e_2)/\dots/e_i$ . It is well known that  $G/T$  is independent of the order of the elements in  $T$ . For example, if  $G$  is a graph with  $V(G) = \{u, v\}$  and parallel edges  $e_1 = \{u, v\}$ ,  $e_2 = \{u, v\}$ , then  $G/e_1$  has one vertex and a loop, while  $G/\{e_1, e_2\}$  has one vertex and no edges. (By identifying  $u$  and  $v$  in  $G$  we obtain a graph with one vertex and two loops.)

For  $e \in E(G)$ , the *deletion*  $G \setminus e$  is a graph  $G'$  with  $V(G') = V(G)$  and  $E(G') = E(G) - \{e\}$ . For a set  $T \subseteq E(G)$ , the deletion  $G \setminus T$  is the graph with vertex set of  $G$  and edge set  $E(G) - T$ .

A *minor* of  $G$  is a graph  $(G \setminus T_1)/T_2$  for some  $T_1 \subseteq E(G)$  and  $T_2 \subseteq E(G \setminus T_1)$ .

A *graph union*  $G_1 \cup G_2$  of the graphs  $G_1$  and  $G_2$  is the graph with vertices in  $V(G_1) \cup V(G_2)$  and edges in  $E(G_1) \cup E(G_2)$ .

A *bridge* is an edge of  $G$  not contained in any cycle of  $G$ .

If  $G$  has an empty edge set then we set the *Tutte polynomial*  $t(G; x, y)$  or  $t(G)$  of  $G$  to be 1. Otherwise we have for any  $e \in E(G)$

**(R1)**  $t(G) = t(G \setminus e) + t(G/e)$  if  $e$  is not a loop or a bridge,

**(R2)**  $t(G) = x t(G \setminus e)$  if  $e$  is a bridge,

**(R3)**  $t(G) = y t(G \setminus e)$  if  $e$  is a loop.

It can be shown that the Tutte polynomial is well-defined [11].  $t(G; x, y)$  is a 2-variable polynomial in  $x, y$  with nonnegative coefficients.

Recall that a *partition*  $P(Y)$  of a finite set  $Y$  is a set  $\{Y_i \mid i \in \{1, \dots, k\}\}$  of disjoint, nonempty subsets of  $Y$ , such that  $Y = \bigcup_{i=1}^k Y_i$ , for some  $k \in \{1, \dots, |Y|\}$ . The subsets  $Y_i$  are called *blocks* of  $P(Y)$  and  $|P(Y)| = k$  denotes their number. A partition  $P_2(Y)$  is a *refinement* of a partition  $P_1(Y)$  if each block of  $P_1(Y)$  is a union of blocks of  $P_2(Y)$ . The set  $\Gamma(Y)$  of all partitions of  $Y$  may be ordered by the following relation  $\prec : P_1(Y) \prec P_2(Y)$  if and only if  $P_2(Y)$  is a refinement of  $P_1(Y)$ . Then  $(\Gamma(Y), \prec)$  is a lattice, called the *partition lattice* of  $Y$ . Obviously, the maximal element of  $(\Gamma(Y), \prec)$  (the top of the lattice) is the partition of  $Y$  consisting of  $|Y|$  singleton blocks. We denote this partition as  $P^1(Y)$ .

For two partitions  $P_1(Y), P_2(Y)$  of  $Y$  let  $P_1(Y) \wedge P_2(Y)$  denote their meet in the partition lattice. For two sets  $X$  and  $Y$  and their partitions  $P(X)$  and  $P(Y)$  we extend the notion of the meet  $P(X) \wedge P(Y)$  to the case  $X \neq Y$  in the following way. Let  $C = X \cup Y$ . Let  $P_X(C)$  be a partition of  $C$  such that every  $a \in C - X$  is in a block which has  $a$  as the only element. Furthermore,  $P_X(C)$  contains all blocks of  $P(X)$ . Analogously,  $P_Y(C)$  is the partition of  $C$  such that  $P_Y(C)$  contains all blocks of  $P(Y)$  and every  $a \in C - Y$  is in a block of  $P_Y(C)$  containing only  $a$ . Then  $P(X) \wedge P(Y)$  is defined as the usual meet  $P_X(C) \wedge P_Y(C)$  in the partition lattice of  $C$ . For example, if  $X \cap Y = \emptyset$ , then  $P(X) \wedge P(Y) = P^1(C)$ .

A *restriction* of a partition  $P(Y)$  to a set  $C$  is a partition  $P'(C \cap Y)$  of the set  $C \cap Y$  with the following property. The blocks of  $P'(C \cap Y)$  are nonempty sets  $C \cap Y'$ , where  $Y'$  is a block of  $P(Y)$ .

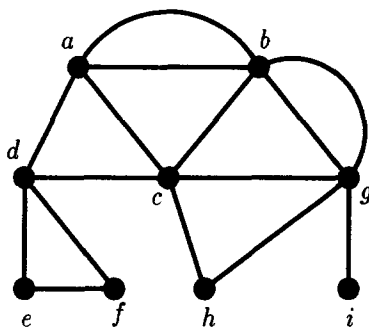
Finally, let  $s(r)$  be the *Bell number*, i.e. the total number of partitions of a set with  $r$  elements.

A *tree decomposition* of an undirected graph  $G$  is a pair  $(T, \mathcal{U})$ , where  $T = (I, F)$  is a tree and  $\mathcal{U} = \{X_i \mid i \in I\}$  is a family of subsets of  $V(G)$ , one for each node in  $T$ , such that

- $\bigcup_{i \in I} X_i = V(G)$ ,
- for all  $\{v, w\} \in E(G)$ , there exists an  $i \in I$  such that  $v \in X_i$  and  $w \in X_i$ ,
- for all  $i_1, i_2, i_3 \in I$ , if  $i_2$  is on the path from  $i_1$  to  $i_3$  in  $T$ , then  $X_{i_1} \cap X_{i_3} \subseteq X_{i_2}$ .

The *width* of a tree decomposition is  $\max_{i \in I} |X_i| - 1$ . The *treewidth* of a graph  $G$  is the minimum width over all possible tree decompositions of  $G$ .

We consider mostly binary, rooted decomposition trees. Following [9], we say that a tree  $T = (I, F)$  is *binary* if each node has at most two sons (note that this is a non-standard use).  $T$  is *rooted* if it has a unique node called *root*. For such a tree and

Fig. 1. An example graph  $G$ .

$i \in I$  we write  $ls(i) \in I$  for the left son of  $i$  and  $rs(i) \in I$  for the right son of  $i$ . We also assume that every node of  $T$  is either a leaf or has at least a left son.

Besides a rooted tree decomposition  $(T, \mathcal{U})$  of  $G$  our algorithm needs a partition  $\{E_i \mid i \in I\}$  of  $E(G)$  defined as follows. For  $i \in I$  let  $E_i$  be a set of edges of  $G$  such that if  $\{v, w\} \in E_i$  then  $v, w \in X_i$ . By definition of  $(T, \mathcal{U})$  such a partition must exist. As our algorithm works for any such partition, we still denote the tree decomposition together with the partition of  $E(G)$  by  $(T, \mathcal{U})$ , but we keep in mind that  $E_1, \dots, E_{|I|}$  are fixed.

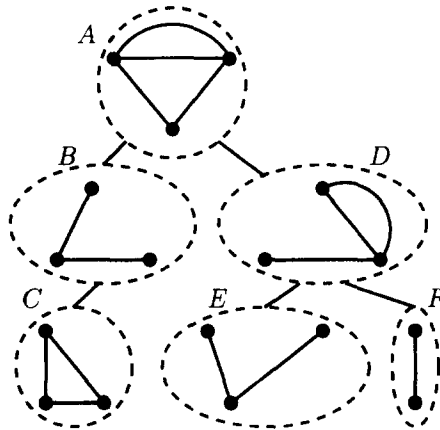
In the following, let  $G$  and its tree decomposition  $(T, \mathcal{U})$  be fixed. For  $i \in I$ , a set  $Y \subseteq V(G)$ , and a partition  $P(Y)$  of  $Y$  let  $G(i, P(Y))$  be the graph obtained in the following way. Let  $G'$  be the graph with  $V(G') = X_i$ ,  $E(G') = E_i$  and let  $P'(Y \cap X_i)$  be the restriction of  $P(Y)$  to  $Y \cap X_i$ . Then  $G(i, P(Y))$  is obtained from  $G'$  by identifying all vertices in each block of the partition  $P'(Y \cap X_i)$ . For example,  $G(i, P^1(Y)) = G'$  for any  $Y \subseteq V(G)$ .

For a node  $i \in I$ , a set  $Y \subseteq V(G)$ , and a partition  $P(Y)$  we define the graph  $\text{sub}G(i, P(Y))$  as follows. Let  $G'$  be the graph union  $\bigcup_j G(j, P^1(X_j))$ , where  $j$  is a descendant of  $i$  (which includes the case  $j = i$ ). Let  $P'(Y \cap V(G'))$  be the restriction of  $P(Y)$  to the vertex set  $Y \cap V(G')$ . Then  $\text{sub}G(i, P(Y))$  is obtained from  $G'$  by identifying all vertices in each block of the partition  $P'(Y \cap V(G'))$ .

For example, for the decomposition tree in Fig. 2 (where each dashed ellipse represents a node of  $T$ ) the graph  $\text{sub}G(A, P^1(\{a, b, c\}))$  is the example graph  $G$  from the Fig. 1. The graph  $\text{sub}G(B, P^1(\{a, c, d\}))$  has vertices  $a, c, d, e, f$  and edges  $\{a, d\}$ ,  $\{c, d\}$ ,  $\{e, d\}$ ,  $\{e, f\}$ ,  $\{f, d\}$ . The graphs  $\text{sub}G(i, P(Y))$ ,  $i \in I$ , are not necessary as a data structure for our algorithm, but they facilitate proofs and explanations.

Next, we explain briefly an algorithm to compute the Tutte polynomial of a graph. We will also argue, why it is a polynomial-time algorithm.

For a fixed  $k$  and a given graph  $G$  we first test in linear time using an algorithm of Bodlaender [7] whether  $G$  has treewidth at most  $k$ . If this is the case, we obtain a tree decomposition of  $G$  of width at most  $k$  as a by-product. Then, applying an algorithm given in [9], we compute in linear time a binary, rooted tree decomposition of  $G$  of

Fig. 2. A tree decomposition of  $G$ .

depth at most  $2 \lceil \log_{5/4}(2n) \rceil$  and width at most  $r = 3k + 2$ . The details of these steps are given in Section 4.

At this stage Algorithm 1 described in Section 5 is applied. At the heart of Algorithm 1 lies the recursive procedure  $TP$ . If  $A$  is the root of a decomposition tree of  $G$ , then the recursive call  $TP(A, P^1(X_A))$  of the procedure  $TP$  yields the Tutte polynomial of  $G$ .

We outline how  $TP$  works.  $TP$  called with parameters  $(i, P(Y))$  computes the Tutte polynomial of the graph  $\text{sub}G(i, P(Y))$ , where  $i \in I$  and  $P(Y)$  is a partition of  $Y \subseteq V(G)$ . Assume that  $i$  has both sons in the decomposition tree of  $G$  (the two other cases are simpler). Put  $G' = \text{sub}G(rs(i), P(Y))$  and  $G'' = \text{sub}G(ls(i), P(Y)) \cup G(i, P(Y))$ . In  $TP$  a so-called splitting formula is applied to  $G'$  and  $G''$ . The output of this formula is the Tutte polynomial of  $\text{sub}G(i, P(Y))$ . The input of the splitting formula are (essentially) the Tutte polynomials of some graphs easily obtained from  $G'$  and the Tutte polynomials of analogous graphs obtained from  $G''$ . These Tutte polynomials are computed by recursive calls of the procedure  $TP$ .

Each call of  $TP$  invokes at most  $2s(r + 1)$  further recursive calls of  $TP$ , where  $r = 3k + 2$ . The maximal depth of the recursion is  $2 \lceil \log_{5/4}(2n) \rceil$ . Hence, the total number of calls of  $TP$  is polynomial in  $n$ . Furthermore, each call of the procedure  $TP$  requires time  $O(n^2)$ , if we exclude the time for further recursive calls of  $TP$ . It follows that the running time of Algorithm 1 is polynomial in  $n$ .

### 3. The splitting formulas

Let  $K$  and  $H$  be two graphs with  $E(K) \cap E(H) = \emptyset$  and let  $G = K \cup H$  be their graph union. We call the set  $U = V(K) \cap V(H)$  the *separator* of  $K$  and  $H$ . Put  $r = |U|$ .

It is well known that for  $r = 1$  we obtain the Tutte polynomial of  $G$  by multiplying  $t(K; x, y)$  by  $t(H; x, y)$ . For fixed  $r \geq 2$  Negami has shown in [15] how to compute the Tutte polynomial of  $G$  using as input the Tutte polynomials (together with the numbers of connected components) of certain graphs easily obtained from  $K$  and from  $H$ . The algorithm, called a *splitting formula*, has polynomial running time in the size of  $G$ .

Let  $r$  be fixed and at least 2. Let  $(\Gamma(U), \prec)$  be the partition lattice of  $U$  (where  $\prec$  is the relation defined in Section 2). Recall that  $s(r)$  is the number of partitions in  $\Gamma(U)$ . Clearly,  $(\Gamma(U), \prec)$  depends only on  $r$  and not on  $G$ . We index the elements of  $\Gamma(U)$  in such a way that for  $P_i, P_j \in \Gamma(U)$  the relation  $P_j \prec P_i$  implies  $j \leq i$ .

Let  $T_r$  be the matrix whose  $(i, j)$ -entry is  $t^{|P_i \wedge P_j|}$ , where  $t$  is a variable (indeterminate). According to [15], the inverse  $T_r^{-1}$  of  $T_r$  exists. We define  $C_r$  as the  $s(r) \times s(r)$ -matrix whose  $(i, j)$ -entry is  $(y-1)^{|P_i|+|P_j|-r} B_{ij}$ , where  $B_{ij}$  is the  $(i, j)$ -entry of  $T_r^{-1}$  with  $t$  replaced by  $(x-1)(y-1)$ . It is not hard to see that  $C_r$  depends only on  $r$  and on the indexing of the partitions in  $\Gamma(U)$ .

For a partition  $P \in \Gamma(U)$  we write  $K//P$  for the graph obtained from  $K$  by identifying each subset of vertices in  $U$  being in the same partition of  $P$ . The graph  $H//P'$  for a partition  $P' \in \Gamma(U)$  is defined analogously. Negami proved:

**Theorem 1** (Splitting formula, Negami [15]). *The Tutte polynomial of  $G$  is given by the splitting formula*

$$t(G; x, y) = (x-1)^{-c(G)} \mathbf{k}_r C_r \mathbf{h}_r^T,$$

where

$$\mathbf{k}_r = [(x-1)^{c(K//P_1)} t(K//P_1; x, y), \dots, (x-1)^{c(K//P_{s(r)})} t(K//P_{s(r)}; x, y)]$$

and

$$\mathbf{h}_r = [(x-1)^{c(H//P_1)} t(H//P_1; x, y), \dots, (x-1)^{c(H//P_{s(r)})} t(H//P_{s(r)}; x, y)].$$

We say that the splitting formula is applied to the graphs  $K$  and  $H$ . If the vectors  $\mathbf{k}_r$  and  $\mathbf{h}_r$  are considered as input then obviously the only time-consuming operations needed to obtain the Tutte polynomial of  $G$  are two matrix multiplications. The entries of the matrices are polynomials.

We give now an example for  $r = 3$ . For  $U = \{u_1, u_2, u_3\}$ , let us denote the elements of  $\Gamma(U)$  as  $P_1 = (u_1, u_2, u_3)$ ,  $P_2 = (u_1; u_2, u_3)$ ,  $P_3 = (u_2; u_1, u_3)$ ,  $P_4 = (u_3; u_1, u_2)$ ,  $P_5 = (u_1; u_2; u_3)$ . Then  $T_3$  turns out to be

$$T_3 = \begin{bmatrix} t & t & t & t & t \\ t & t^2 & t & t & t^2 \\ t & t & t^2 & t & t^2 \\ t & t & t & t^2 & t^2 \\ t & t^2 & t^2 & t^2 & t^3 \end{bmatrix}$$

and the matrix  $C_3$  is

$$C_3 = d \begin{bmatrix} (x-1)^2 & 1-x & 1-x & 1-x & 2 \\ 1-x & xy-x-y & 1 & 1 & 1-y \\ 1-x & 1 & xy-x-y & 1 & 1-y \\ 1-x & 1 & 1 & xy-x-y & 1-y \\ 2 & 1-y & 1-y & 1-y & (y-1)^2 \end{bmatrix},$$

where

$$d = \frac{1}{(x-1)(xy-x-y-1)(xy-x-y)}.$$

#### 4. The tree decomposition algorithm

In this section we summarize the results of Bodlaender and Hagerup, obtaining Corollary 4.

**Lemma 2** (Bodlaender and Hagerup [9]). *Let  $k$  be a constant. Given a tree decomposition of width  $k$  of a graph  $G$  on  $n$  vertices, we can compute a rooted, binary tree decomposition of  $G$  of depth at most  $2 \lceil \log_{5/4}(2n) \rceil$  and width at most  $3k+2$  in time  $O(n)$  (using a sequential algorithm).*

**Proof.** The algorithm for the problem is given in [6, Theorem 4.1 and 4.2] and improved in [9]. It is shown to solve the problem with  $O(n)$  operations in time  $O(\log n)$  on an EREW PRAM. As the processor allocation is no problem (see [14]), we can apply Brent's scheduling principle [10]: a parallel algorithm requiring  $w(n)$  operations and  $t(n)$  time can be simulated using  $p$  processors in time  $w(n)/p + t(n)$ . Thus, a sequential algorithm for this problem will require  $O(n)$  time.  $\square$

If an instance is given by a graph  $G$  and an integer  $k$ , then the problem of determining if  $G$  has treewidth at most  $k$  is NP-complete. On the other hand we have the following result of Bodlaender [7].

**Theorem 3** (Bodlaender [7]). *For all positive integers  $k$  there exists a linear-time algorithm that tests whether a given graph  $G=(V,E)$  has treewidth at most  $k$ , and if so, outputs a tree decomposition of  $G$  of width at most  $k$ .*

Combining Lemma 2 and Theorem 3 we obtain the following result.

**Corollary 4.** *For all positive integers  $k$  there is a linear-time algorithm, which tests whether a given graph  $G$  on  $n$  vertices has treewidth at most  $k$ , and if so, outputs a binary, rooted tree decomposition of  $G$  of depth at most  $2 \lceil \log_{5/4}(2n) \rceil$  and width at most  $3k+2$ .*

## 5. The main algorithm

Let  $k$  be a fixed, positive integer. For an input graph  $G$ , we apply to  $G$  the algorithm of Corollary 4. If the treewidth of  $G$  is larger than  $k$ , then we stop. Otherwise, we may assume that a tree decomposition  $(T, \mathcal{U})$  of  $G$  has been computed and that it has width at most  $3k + 2$ . Moreover,  $T$  is binary and has depth at most  $2 \lceil \log_{5/4}(2n) \rceil$ . Recall that the tree  $T$  has node set  $I$  and the edge set  $F$ , that  $\mathcal{U} = \{X_i \mid i \in I\}$  is a family of subsets of  $V(G)$ , and that  $\{E_i \mid i \in I\}$  is a fixed partition of  $E(G)$ . Put  $r = 3k + 2$ .

### Algorithm 1

*Input:* A rooted, binary tree decomposition of a graph  $G$  of width at most  $r$ .

*Output:* The Tutte polynomial of  $G$ .

*Actions:* Call the recursive procedure  $TP$  by  $TP(A, P^1(X_A))$ , where  $A$  is the root of the decomposition tree  $T$ .

**proc**  $TP$ -one-son-or-leaf( $i, P(Y)$ )

**if**  $i = \text{leaf of } T$

**then return** the Tutte polynomial  $t(G(i, P(Y)); x, y)$ , computed using the rules R1, R2 and R3;

**else** ( $i$  has a left son  $\text{ls}(i)$ ).

$C_i := X_i \cap X_{\text{ls}(i)}$  and  $p := |C_i|$ ;

compute  $c(\text{sub}G(i, P(Y)))$ ;

**for** each partition  $R$  of  $C_i$  **do**

compute  $t(G(i, P(Y) \wedge R); x, y)$  using the rules R1, R2, R3;

compute  $c(G(i, P(Y) \wedge R))$ ;

**od**

regard the result of the last loop as a vector  $k_p$  for the splitting formula (with graph  $K = G(i, P(Y))$ );

**for** each partition  $R$  of  $C_i$  **do** (Loop A)

**call**  $TP(\text{ls}(i), P(Y) \wedge R; x, y)$ ;

compute  $c(\text{sub}G(\text{ls}(i), P(Y) \wedge R))$ ;

**od**

regard the result of the last loop as a vector  $h_p$  for the splitting formula (with graph  $H = \text{sub}G(\text{ls}(i), P(Y))$ );

**return** result of a splitting formula with input  $k_p, h_p$  and  $c(\text{sub}G(i, P(Y)))$ ;

**fi.**

**proc**  $TP(i, P(Y))$

**if** right son of  $i$  does not exists

**then** result := **call**  $TP$ -one-son-or-leaf ( $i, P(Y)$ );

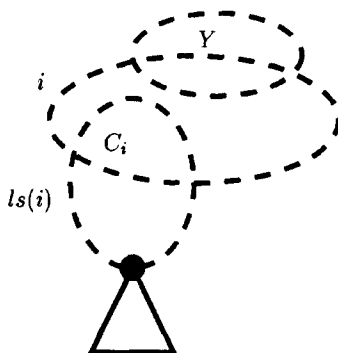
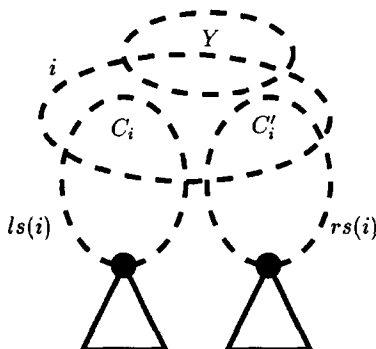
**return** result;



**else** $C'_i := X_i \cap X_{rs(i)}$  and  $p := |C'_i|$ ;compute  $c(\text{sub}G(i, P(Y)))$ ;**for** each partition  $R'$  of  $C'_i$  **do** (Loop B)**call**  $TP(rs(i), P(Y) \wedge R')$  in order to compute $t(\text{sub}G(rs(i), P(Y) \wedge R'); x, y)$ ;compute  $c(\text{sub}G(rs(i), P(Y) \wedge R'))$ ;**od**regard the result of the last loop as a vector  $k_p$  for the splittingformula (with graph  $K = \text{sub}G(rs(i), P(Y))$ );remove the edge  $\{i, rs(i)\}$  from  $T$ ;(Now  $i$  has only the left son).**for** each partition  $R'$  of  $C'_i$  **do** (Loop C)**call**  $TP\text{-one-son-or-leaf}(i, P(Y) \wedge R')$  in order to compute $t(\text{sub}G(i, P(Y) \wedge R'); x, y)$  (which equals $t(\text{sub}G(ls(i), P(Y) \wedge R') \cup G(i, P(Y) \wedge R'); x, y)$ because  $\{i, rs(i)\}$  is removed from  $T$ );compute  $c(\text{sub}G(i, P(Y) \wedge R'))$ ;**od**regard the result of the last loop as a vector  $h_p$  for the splittingformula (with graph  $H = \text{sub}G(ls(i), P(Y)) \cup G(i, P(Y))$ );**return** result of a splitting formula with input  $k_p, h_p$ and  $c(\text{sub}G(i, P(Y)))$ ;**fi.**

Let us describe Algorithm 1. For an  $i \in I$  and  $Y \subseteq V(G)$  a call  $TP(i, P(Y))$  or  $TP\text{-one-son-or-leaf}(i, P(Y))$  yields the Tutte polynomial of the graph  $\text{sub}G(i, P(Y))$ , but the procedure  $TP\text{-one-son-or-leaf}$  will be called only if node  $i$  of  $T$  has no right son. (The procedure  $TP\text{-one-son-or-leaf}$  has been introduced to shorten the listing. It can be embedded in the procedure  $TP$ .)

Assume that  $i \in I$  has no right son (this can happen even if  $i$  has had a right son before the first call of  $TP$ , as  $T$  can change during the computation of the algorithm). Then  $TP\text{-one-son-or-leaf}$  is called. If  $i$  is a leaf of  $T$ , then we can compute the Tutte polynomial of  $G(i, P(Y))$  using the rules R1, R2, and R3 defined in Section 2. Otherwise we have the situation as in Fig. 3. Then a splitting formula is applied to the graphs  $\text{sub}G(ls(i), P(Y))$  and  $G(i, P(Y))$ . For each partition  $R$  of the separator  $C_i = X_i \cap X_{ls(i)}$  the splitting formula requires as input both the number of connected components and the Tutte polynomial of the graph  $G(i, P(Y) \wedge R)$ . The Tutte polynomials of these graphs are computed using the rules R1, R2 and R3. As further input for the splitting formula we need the number of connected components and the Tutte polynomials of the graphs  $\text{sub}G(ls(i), P(Y) \wedge R)$  for each partition  $R$  of  $C_i$ . There are at most  $s(|C_i|)$  such graphs. The Tutte polynomials of them are computed recursively

Fig. 3. Node  $i$  has one son.Fig. 4. Node  $i$  has two sons.

calling  $TP(ls(i), P(Y) \wedge R)$  in each case. After the computation of  $c(\text{sub}G(i, P(Y)))$  we can apply the splitting formula.

If the case of Fig. 4 occurs, i.e.  $i$  has both sons, then the **else**-branch of the procedure  $TP$  is executed. Here we apply the splitting formula to the graphs  $\text{sub}G(rs(i), P(Y))$  and  $\text{sub}G(ls(i), P(Y)) \cup G(i, P(Y))$ . The connecting intersection is  $C'_i = (X_i \cup X_{ls(i)}) \cap X_{rs(i)} = X_i \cap X_{rs(i)}$ . We obtain the input for the splitting formula in following way. For each partition  $R'$  of  $C'_i$  we compute the number of connected components and the Tutte polynomial of the graph  $\text{sub}G(rs(i), P(Y) \wedge R')$ . The Tutte polynomials are computed by recursive calls  $TP(rs(i), P(Y) \wedge R')$ . Then, for each partition  $R'$  of  $C'_i$  we compute the number of connected components and the Tutte polynomial of the graph  $\text{sub}G(ls(i), P(Y) \wedge R') \cup G(i, P(Y) \wedge R')$ . To achieve this we remove the edge  $\{i, rs(i)\}$  (i.e. cut off the right son of  $i$  and its subtree in  $T$ ). Then the Tutte polynomial of  $\text{sub}G(ls(i), P(Y) \wedge R') \cup G(i, P(Y) \wedge R')$  can be computed by the recursive call  $TP\text{-one-son-or-leaf}(i, P(Y) \wedge R')$ . In the last step the number of connected components of  $\text{sub}G(i, P(Y))$  is computed and the splitting formula is applied.

Algorithm 1 requires some preprocessing. Especially, for each  $i = 2, \dots, r$  the partition lattice of a set of cardinality  $i$  as well as the matrix  $C_i$  of the splitting formula must be computed.

Clearly, the preprocessing has not a polynomial running time in  $r$  (unless  $P = \#P$ ), otherwise we would have a polynomial algorithm for the computation of the Tutte polynomial of any graph  $G$ . The preprocessing seriously limits the practical applicability of the algorithm. Even for small treewidth  $k$  of the input graph the numbers  $s(r+1)$  are large (see Table 1). For graphs with treewidth at most 3 we already have to compute and store the matrices  $C_i$  of the splitting formulas for  $i = 2, \dots, 9$ , where  $C_9$  is a  $8427194 \times 8427194$ -matrix. (Recall that  $C_9$  is an inverse of a matrix of the same size!)

**Theorem 5.** *Algorithm 1 computes the Tutte polynomial of an input graph  $G$  of treewidth at most  $k$ .*

**Proof.** If  $A$  is the root of  $T$ , then  $\text{sub}G(A, P^1(X_A)) = G$ . So it is sufficient to show that for any  $i \in I$  the call  $TP(i, P(Y))$  terminates and computes the Tutte polynomial of  $\text{sub}G(i, P(Y))$ . Here  $P(Y)$  is a partition of  $Y \subseteq V(G)$ . We show this by induction on the height  $h(i)$  of  $i \in I$  in  $T$ , i.e. the depth of the subtree of  $T$  whose root is  $i$ . By convention, the leaves of  $T$  have height 0. We use the notations as in Algorithm 1. If  $h(i) = 0$  then  $\text{sub}G(i, P(Y)) = G(i, P(Y))$  and  $t(\text{sub}G(i, P(Y)); x, y)$  is computed using the rules R1, R2, and R3.

For the induction step it is a tedious but routine task to see that a splitting formula is correctly applied to graphs  $G(i, P(Y))$  and  $\text{sub}G(\text{ls}(i), P(Y))$  (if  $i$  has no right son in  $T$ , procedure *TP-one-son-or-leaf*) or to graphs  $\text{sub}G(\text{rs}(i), P(Y))$  and  $\text{sub}G(\text{ls}(i), P(Y)) \cup G(i, P(Y))$  (if  $i$  has both sons, procedure *TP*). Part of the input for the splitting formula is computed by recursive calls of the procedures *TP* and *TP-one-son-or-leaf*. By induction assumption these calls give the correct Tutte polynomials, as  $h(i') < h(i)$  for any further recursive call  $TP(i', P'(Y'))$  or *TP-one-son-or-leaf* ( $i', P'(Y')$ ), where  $i' \in I$  and  $P'(Y')$  is a partition of  $Y' \subseteq V(G)$ . Also the correctness of the remaining input of the splitting formula (the number of connected components of certain graphs) is clear. We conclude by Theorem 1 that  $TP(i, P(Y))$  terminates and gives the Tutte polynomial of  $\text{sub}G(i, P(Y))$ .  $\square$

## 6. The running time of the main algorithm

We obtain an upper bound on the running time of Algorithm 1 in the following way. First, we bound the number of calls of the procedure *TP*. Then we calculate the maximal time to complete a single call of *TP* (without the time required for further recursive calls).

To obtain a better bound on the running time and make the analysis simpler we assume that the procedure *TP-one-son-or-leaf* called in the loop of *TP* is embedded

into the body of the procedure  $TP$ . For every node  $i \in I$  with two sons in such a modified procedure  $TP$  called with parameters  $(i, P(Y))$  a double-nested loop is executed. Loop C is the outer loop and Loop A the inner loop. In this double-nested loop, for each partition  $R'$  (the outer loop index variable) of  $C'_i = X_i \cap X_{rs(i)}$  the inner loop goes over all partitions  $R$  (the inner loop index variable) of  $C_i = X_i \cap X_{ls(i)}$ . Observe that the double-nested loop makes no more than  $s(r+1)$  recursive calls of  $TP$ , because the number of all meets  $R \wedge R'$  is at most the number of all partitions of  $X_i$ . As the other loops of  $TP$  (loops A and B) make at most  $s(r+1)$  recursive calls of  $TP$ , this procedure makes in total no more than  $2s(r+1)$  recursive calls of itself.

We introduce a *splitting tree*  $\tilde{S} = (M, Q)$ . The set of nodes  $M$  of  $\tilde{S}$  contains (certain) pairs  $(i, P(Y))$ , where  $i \in I$  and  $P(Y)$  is a partition of a  $Y \subseteq V(G)$ .  $\tilde{S}$  has an edge  $e \in Q$  between nodes  $(i_1, P_1(Y_1))$  and  $(i_2, P_2(Y_2))$  if  $(i_1, P_1(Y_1)) \neq (i_2, P_2(Y_2))$  and if a (modified) procedure  $TP$  called with arguments  $(i_1, P_1(Y_1))$  makes the call  $TP(i_2, P_2(Y_2))$  during the computation. The nodes of  $\tilde{S}$  are defined as the endpoints of the edges just described. Obviously  $\tilde{S}$  is a tree. Its root is  $(A, P^1(X_A))$ , where  $A$  is a root of  $T$ , the decomposition tree. Furthermore, each node of  $\tilde{S}$  has at most  $2s(r+1)$  sons, as shown in the previous paragraph. It is not hard to see that  $\tilde{S}$  has depth of  $T$ , i.e. its depth is at most  $2 \lceil \log_{5/4}(2n) \rceil$ .

Let us put  $c = 2s(r+1)$ . We have:

**Lemma 6.** *The number of nodes of the tree  $\tilde{S}$  is at most*

$$(c-1)^{-1} c^3 (2n)^{2 \log_2(c) / \log_2(5/4)}.$$

**Proof.** The depth of  $\tilde{S}$  is at most  $2 \lceil \log_{5/4}(2n) \rceil$  and so we have:

$$\begin{aligned} |M| &\leq 1 + c + (c)^2 + \dots + (c)^{2 \lceil \log_{5/4}(2n) \rceil} = (c-1)^{-1} (c^{2 \lceil \log_{5/4}(2n) \rceil + 1} - 1) \\ &< (c-1)^{-1} c^{2 \log_{5/4}(2n) + 3} = (c-1)^{-1} c^3 \left( \left( \frac{5}{4} \right)^{\log_{5/4}(c)} \right)^{2 \log_{5/4}(2n)} \\ &= (c-1)^{-1} c^3 \left( \frac{5}{4} \right)^{\log_{5/4}(c) 2 \log_{5/4}(2n)} = (c-1)^{-1} c^3 (2n)^{2 \log_{5/4}(c)}. \end{aligned}$$

The identity  $\log_{5/4}(c) = \log_2(c) / \log_2(5/4)$  completes the proof.  $\square$

The following lemma bounds from above the running time of Algorithm 1 corresponding to a single node of  $\tilde{S}$ .

**Lemma 7.** *For sufficiently large  $n$ , the time required to complete a single call of the procedure  $TP$  without the time required for further recursive calls is bounded by*

$$a_2 n^2 + (a_1 2^{\binom{r+1}{2}} + a_3) s(r+1) + a_4 (s(r+1))^4,$$

where  $a_1, a_2, a_3$  and  $a_4$  are positive constants (independent of  $r$  and of  $n$ ).

**Proof.** For a node  $(i, P(Y)) \in \bar{S}$ , we have to perform one or more of the following steps:

- (1) Compute the Tutte polynomials of the graphs  $G' = G(i, P(Y) \wedge R')$  using the rules R1, R2, and R3, where  $R'$  ranges over all partitions of a subset of  $X_i$ .
- (2) For each partition  $R'$  of  $X_i$  find the number of connected components of the graphs  $\text{sub}G(j, R' \wedge P(Y))$  for each  $j = \text{ls}(i)$ ,  $j = \text{rs}(i)$  (if applicable) and  $j = i$  (if applicable). Furthermore find the number of connected components of the graph  $\text{sub}G(i, P(Y))$  and of the graphs  $G(i, R' \wedge P(Y))$ , where  $R'$  ranges over all partitions of a subset of  $X_i$ .
- (3) Apply at most  $s(r+1)+1$  many times splitting formulas and execute the remaining operations in the procedure body such as comparisons, loop initializations, etc.

To (1): For a fixed partition  $R'$  of a subset  $W$  of  $X_i$  we estimate the time to compute  $t(G'; x, y)$ . Each set  $\{e_1, \dots, e_m\} \subseteq E_i$ ,  $m \geq 2$  of parallel (non-loop) edges can be regarded as a single edge because of the following generalization of the rule R1, which can be easily shown by induction:

$$t(G'; x, y) = t(G' \setminus \{e_2, \dots, e_m\}; x, y) + (y + \dots + y^{m-1})t(G'/e_1 \setminus \{e_2, \dots, e_m\}; x, y).$$

If the rule R1 is applicable to  $e_1$  in  $t(G' \setminus \{e_2, \dots, e_m\}; x, y)$ , then we obtain

$$\begin{aligned} t(G'; x, y) &= t(G' \setminus \{e_1, \dots, e_m\}; x, y) \\ &\quad + (1 + y + \dots, y^{m-1})t(G'/e_1 \setminus \{e_2, \dots, e_m\}; x, y). \end{aligned}$$

Otherwise, R2 or R3 can be applied to  $e_1$  in  $t(G' \setminus \{e_2, \dots, e_m\}; x, y)$ .

Therefore, the time for the computation of  $t(G'; x, y)$  depends only on the number of edges in the underlying simple graph  $G''$  of  $G'$ . The graph  $G''$  has at most  $\binom{|V(G')|}{2}$  edges. As  $|V(G')| \leq r+1$ , the time for computation of  $t(G'; x, y)$  is bounded from above by a constant depending only on  $r$ . Depending on the implementation this constant may vary. If a graph  $G''$  has  $m$  edges, then applying R1 or generalized R1 to an appropriate edge we create two minors of  $G'$  with  $m-1$  edges each. Thus, we have  $2^m$  as a rough upper bound on the number of applications of the rules R1, R2 and R3. Consequently, if each application of a rule R1, R2 or R3 needs constant a time  $a_1$ , then  $a_1 2^{\binom{r+1}{2}}$  is the time for the computation of  $t(G'; x, y)$ .

There are at most  $s(r+1)$  partitions  $R'$  of a subset  $W$  of  $X_i$ , and so the algorithm spends at most the time

$$a_1 2^{\binom{r+1}{2}} s(r+1)$$

applying the rules R1, R2 and R3.

To (2): First, we perform a depth-first search (DFS) on  $\text{sub}G(i, P(Y))$  in order to find its connected components. For each vertex  $v \in X_i$  we store a name of the connected component of  $v$ . A DFS has running time at most  $a_2(|V(G)| + |E(G)|)$  (or  $a_2((|V(G)|)^2)$  if  $\text{sub}G(i, P(Y))$  is not given as an adjacency list), where  $a_2$  is some small constant depending on implementation. Ignoring parallel edges we see that DFS needs time at most  $a_2 n^2$  for sufficiently large  $n$ .

Table 1  
Some characteristics of the main algorithm for small values of  $k$

$k$	2	3	4	5	8
$c$	42294	842 7194	276 591 709	$1.36\,415 \times 10^{12}$	$1.09\,143 \times 10^{21}$
$e(k)$	97	145	197	252	436

For any partition  $R'$  of a subset of  $X_i$  the number of connected components of  $\text{sub}G(j, R' \wedge P(Y))$ ,  $j \in \{\text{ls}(i), i, \text{rs}(i)\}$  can be found using the information stored for each vertex in  $X_i$  in time linear in  $r$ .

Similarly, the time for finding the number of connected components of  $G(i, R' \wedge P(Y))$  for a fixed  $R'$  is linear in  $r$ , if for each vertex  $v \in X_i$  we store a name of the connected component of  $G(i, P(Y))$  containing  $v$ . This information can be obtained by performing a DFS on  $G(i, P(Y))$ , which takes time  $O(r^2)$ .

Hence, the total time to compute the number of connected components of all graphs mentioned in (2) is bounded by

$$a_2 n^2 + a_3 s(r + 1)$$

(for sufficiently large  $n$ ), because we loop at most four times over at most  $s(r + 1)$  partitions of some subsets of  $X_i$  and because  $r^2 \leq s(r + 1)$  for any  $r > 0$ .

To (3): The time for this is dominated by the matrix multiplications of the matrices  $C_m$ ,  $m \leq r + 1$ , of the splitting formulas and it is bounded by  $a_4(s(r + 1))^4$ , where  $a_4$  is a small constant.

We obtain the statement of the lemma by summing up the costs of the operations described in (1)–(3).  $\square$

Combining Lemmas 6 and 7 we can bound the running time of Algorithm 1 from above by

$$(c - 1)^{-1} c^3 (2n)^{2 \log_2(c)/\log_2(5/4)} [a_2 n^2 + (a_1 2^{\binom{r+1}{2}} + a_3) s(r + 1) + a_4 (s(r + 1))^4].$$

We have shown the following lemma.

**Lemma 8.** *Let  $G$  be a graph on  $n$  vertices and  $(T, \mathcal{U})$  a rooted, binary tree decomposition of depth at most  $2 \lceil \log_{5/4}(2n) \rceil$  of width at most  $r$ . Then Algorithm 1 computes the Tutte polynomial of  $G$  in time  $O((2n)^{2+2 \log_2(c)/\log_2(5/4)})$ , where  $c$  is twice the number of partitions of a set with  $r + 1$  elements.*

We obtain the main result of this paper by the last lemma and Corollary 4.

**Theorem 9.** *For each positive integer  $k$  there is an algorithm which decides in linear time if a given graph  $G$  on  $n$  vertices has treewidth at most  $k$  and if so, it computes the Tutte polynomial of  $G$  in total time  $O(n^{2+7 \log_2 c})$ , where  $c$  is twice the number of partitions of a set with  $3k + 3$  elements.*

Table 1 gives the values of  $c$  and the exponent  $e(k)$  of the expression  $(2n)^{2+2 \log_2(c)/\log_2(5/4)}$  for some small  $k$ 's.

As a consequence of Theorem 9 the large class of problems which are computable in polynomial time for graphs of bounded treewidth can be expanded by the following problems. The solution to each of them is given by direct specialization of the Tutte polynomial (in some cases multiplied with an easily obtainable factor).

**Corollary 10.** *For any positive integer  $k$  and for each of the following problems there is an algorithm which solves the respective problem in time polynomial in  $n$  for a given graph  $G$  of treewidth at most  $k$ . The problems are to find for  $G$  the*

1. *chromatic polynomial of  $G$  [19];*
2. *number of nowhere zero flows of  $G$  [19];*
3. *(all terminal) reliability of  $G$  [19];*
4. *partition function of the  $q$ -state Potts model of statistical mechanics (for  $q = 2$  it is the partition function of the well-known Ising model) [19];*
5. *partition function of the random cluster model introduced by Fortuin and Kasteleyn [19];*
6. *number of acyclic orientations of  $G$  [13];*
7. *number of acyclic suborientations of  $G$  [12];*
8. *number of initially connected acyclic suborientations of  $G$  [12];*
9. *number of connected subdigraphs of  $G$  [12];*
10. *number of different score vectors associated with an orientation of  $G$  [13];*
11. *number of connected subdigraphs of  $G$  [12];*
12. *number  $W(G, m)$  which denotes the number of pairs  $(A, f)$  such that  $A$  is an acyclic orientation of  $G$  and  $f: V(G) \rightarrow \{1, \dots, m\}$  is a function which holds  $f(u) \geq f(v)$  for every edge of  $G$  directed from  $u$  to  $v$  (for  $m = 1$  this is the number of acyclic orientations of  $G$ ) [13];*
13. *Jones polynomial of an oriented alternating link diagram, where  $G$  is its associated unsigned ‘blackface’ graph [19];*

Many of the listed problems are known to be  $\#P$ -hard for any graph class containing all planar graphs [19].

## Acknowledgements

Dominic Welsh et al. have proved that the Tutte polynomial of a graph of bounded treewidth is polynomial-time computable but never published the result. The author would like to thank him for encouragement to find an another proof.

The author also thanks Jan Arne Telle, University Bergen, for his valuable suggestions on algorithms for graphs of bounded treewidth.

## References

- [1] M. Aigner, *Combinatorial Theory*, Springer, New York, 1979.
- [2] A. Andrzejak, Splitting formulas for Tutte polynomials, *J. Combin. Theory Ser. B* 70 (1997) 346–366.
- [3] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability — a survey, *BIT* 25 (1985) 2–23.
- [4] S. Arnborg, J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs, *J. Algorithms* 12 (2) (1991) 308–340.
- [5] S. Arnborg, A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial  $k$ -trees, *Discrete Appl. Math.* 23 (1) (1989) 11–24.
- [6] H.L. Bodlaender, NC-Algorithms for graphs with small treewidth, in: J. van Leeuwen (Ed.), *WG'88*, Amsterdam, Netherlands, Springer, Berlin, 344, 1989, pp. 1–10.
- [7] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, *SIAM J. Comput.* 25 (1996) 1305–1317.
- [8] H.L. Bodlaender, A tourist guide through treewidth, *Acta Cybernet.* 11 (1–2) (1993) 1–21.
- [9] H.L. Bodlaender, T. Hagerup, Parallel algorithms with optimal speedup for bounded treewidth, *ICALP'95*, Szeged, Hungary, Springer, Berlin, 944, 1995, 268–290.
- [10] R.P. Brent, The parallel evaluation of general arithmetic expressions, *J. Assoc. Comput. Mach.* 21 (1974) 201–206.
- [11] T. Brylawski, J. Oxley, The Tutte polynomial and its applications, *Matroid applications*, *Encycl. Math. Appl.*, vol. 40, 1992, pp. 123–225.
- [12] I.M. Gessel, B.E. Sagan, The Tutte polynomial of a graph, depth-first search, and simplicial complex partitions, *Electronic J. Comb.* 3 (2) *The Foata Festschrift*, 1996.
- [13] F. Jaeger, D.L. Vertigan, D.J.A. Welsh, On the computational complexity of the Jones and Tutte polynomials, *Math. Proc. Camb. Philos. Soc.* 108 (1) (1990) 35–53.
- [14] R.M. Karp, V. Ramachandran, Parallel algorithms for shared-memory machines, in: J. Van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, vol. A: Algorithms and Complexity, MIT Press, Cambridge, MA, 1990, pp. 869–941.
- [15] S. Negami, Polynomial Invariants of Graphs, *Trans. Am. Math. Soc.* 299 (1987) 601–622.
- [16] S. Noble, Evaluating the Tutte Polynomial for Graphs of Bounded Tree-Width, private communication.
- [17] J.G. Oxley, D.J.A. Welsh, Tutte polynomials computable in polynomial time, *Discrete Math.* 109 (1–3) (1992) 185–192.
- [18] V.L. Vertigan, The computational complexity of Tutte invariants for planar graphs, to appear.
- [19] D.J.A. Welsh, *Complexity: Knots, colourings and counting*, London Mathematical Society Lecture Note Series, vol. 186, Cambridge University Press, Cambridge, 1993.
- [20] D.J.A. Welsh, The computational complexity of knot and matroid polynomials, *Discrete Math.* 124 (1–3) (1994) 251–269.